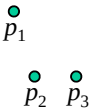
2.5 Reverse nächste Nachbarn Anfragen

– Allgemeines
  - Eigenschaften
    – Benutzer gibt Anfrageobjekt *q* vor
    – Ergebnis enthält alle Objekte, die *q* als nächsten Nachbarn haben
    – Analog: Reverse *k*-nächste Nachbarn
    – Mehrdeutigkeiten (bei NN) entsprechend behandeln
  - Formal
    – Reverse nächste Nachbarn $RNN(q) = \{o \in DB \mid q \in NN(o)\}$
    – Reverse *k*-nächste Nachbarn $RNN(q,k) = \{o \in DB \mid q \in NN(o,k)\}$
  - Anwendungsbeispiel:
    – Standortsuche für neue Filiale (welche Kunden haben die neue Filiale als „nächsten Nachbarn“)

– Zusammenhang zwischen NN und RNN
  - NN ist keine symmetrische Relation
    – y ∈ NN(x) ⇎ x ∈ NN(y)
    – y ∈ NN(*x*) ⇎ y ∈ RNN(*x*)
  - RNN ist ein „eigenständiges Problem



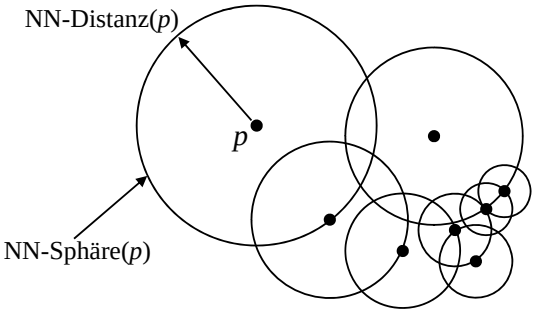
	NN	RNN
p_1	$\{p_2\}$	{}
p_2	$\{p_3\}$	$\{p_3, p_1\}$
p_3	$\{p_2\}$	$\{p_2\}$

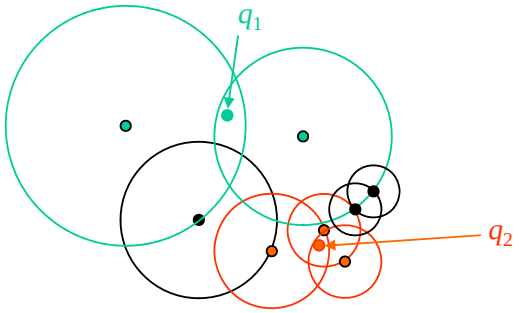
– Basisalgorithmus (sequential scan): nichtdeterministisch

```
RNN-SeqScan(DB, q, k)
    resultSet = ∅;
    FOR i=1 TO n DO
        neighbors = NN-SequScan(DB, DB.getObject(i), k);
        IF q ∈ neighbors THEN
            resultSet.add(DB.getObject(i));
    RETURN resultSet;
```

  - Offensichtliche Verbesserung
    – Statt NN-SequScan einen besseren NN-Algorithmus verwenden

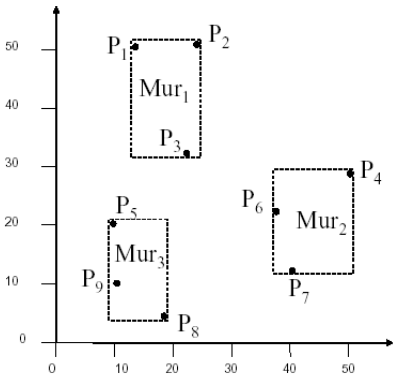
– Weitere Verbesserung [Korn, Muthukrishnan. ACM Int. Conf. Management of Data (SIGMOD), 2000]
  - $p \in RNN(q) \Leftrightarrow dist(p, q) \leq$ NN-Distanz(p)
  - Materialisiere für alle Objekte die NN-Distanz
  - Prüfe, ob $dist(p, q) \leq$ NN-Distanz(p) statt während der Anfrage eine NN-Query für alle Objekte zu berechnen



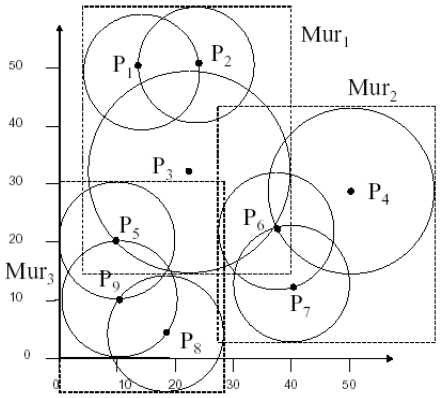


  - Für Vektordaten
    – RNN-Query ist Punktanfrage bzgl. der NN-Sphären der Punkte (vgl. Voronoi-Ansatz zur NN-Query)
    – Speichere NN-Sphären in einem Index für ausgedehnte Objekte (z.B. R-Baum)

– RNN-Baum [Korn, Muthukrishnan. ACM Int. Conf. Management of Data (SIGMOD), 2000]
  - RNN-Queries für Vektordaten
  - Berechne NN-Distanz für alle DB-Punkte
  - Speichere statt Punkte alle NN-Sphären der Punkte in R-Baum



Normaler R-Baum



RNN-Baum

- Algorithmus zur NN-Suche
  - Datenseiten enthalten Kreise, d.h. Objekte der Form (Punkt, Radius)
    - » Punkt = DB-Objekt (Mittelpunkt)
    - » Radius = NN-Distanz(Punkt)

```
RNN-Tree-Search(pa, q,)          // pa = Diskadress z.B. der Wurzel des Indexes
   result = ∅;
   p := pa.loadPage();
   IF p.isDataPage() THEN
      FOR i=0 TO p.size() DO
         IF dist(q, p.getObject(i).Point) ≤ p.getObject(i).Radius THEN
            result := result ∪ getObject(i).Point;

   ELSE                          // p ist Directoryseite
      FOR i=0 TO p.size() DO
         IF MINDIST(q, p.getRegion(i)) = 0 THEN
            result := result ∪ RNN-Tree-Search(p.childPage(i), q);
   RETURN result;
```


  - Vorteil
    - » Sehr gute Performanz (Pruning-Power) bei RNN-Anfragen
  - Nachteile
    - » *k* muss fest vorgegeben sein
    - » Nur für Vektordaten
    - » Hohe Überlappungen der Seitenregionen im Directory führt zu schlechter Performanz bei normalen NN-Anfragen
      Lösung: speichere einen RNN-Baum und einen konventionellen R-Baum
    - » Schlechte Performanz bei Einfügungen und Löschungen
      Beispiel: *Einfügen* von *p*

„Normaler“ Index	Bestimme NN(*p*) und füge Kreis (*p*, NN-Distanz(*p*)) in Index ein
Zusätzlich für RNN-Baum	Bestimme RNN(*p*)
	Erneuere NN-Sphären aller $o \in$ RNN(*p*)
	Erneuere Seitenregionen (von *p* und allen *o*) betroffener Datenseiten
	Erneuere rekursiv Seitenregionen der Vaterseiten

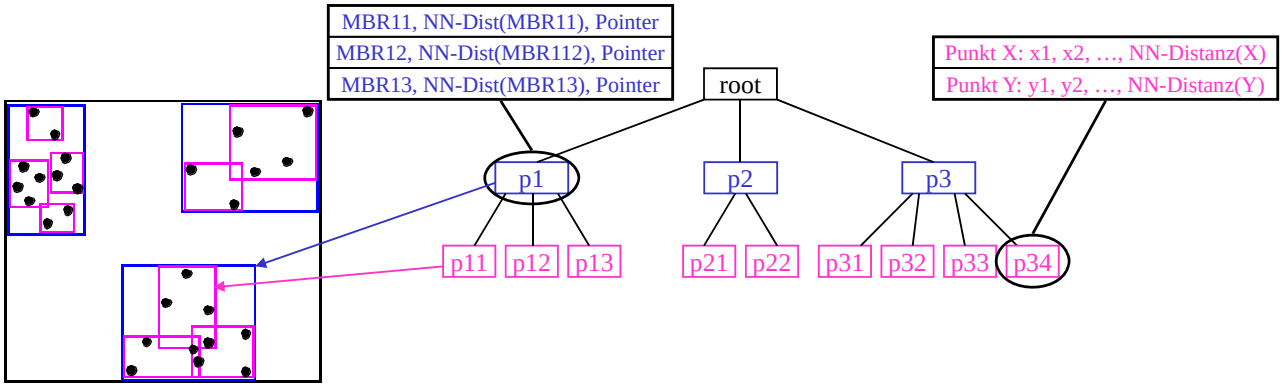
  - Varianten:
    - » Voronoi-Zellen statt NN-Sphären
    - » Andere Verfeinerungsreihenfolge (siehe NN-Algorithmen)

– RdNN-Baum [Yang, Lin. IEEE Int. Conf. Data Engineering (ICDE), 2001]

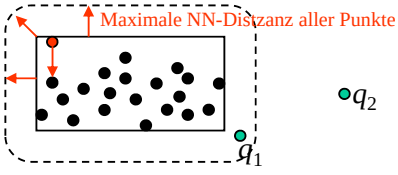
- Prinzip
  - Idee des RNN-Baum, ABER
  - Speichere die DB-Objekte statt NN-Sphären
  - Speichere zu jedem Punkt in der DB seine NN-Distanz
    - » Zu jedem Punkt zusätzlich einen Wert abspeichern
    - » Zu jeder Seitenregion *s* zusätzlich noch den Wert

$$\max_{child \in s} child.getNNDist()$$

abspeichern (Maximum aller NN-Distanzen in den Kinderseiten)



– Ausschluss von Directory-Seiten, wenn MINDIST(q,Seitenregion) größer ist, als die aggregierte NN-Distanz der Seitenregion



Query q_2: Seite kann ausgeschlossen werden; kein Punkt der Seite kann q_2 als NN haben

Query q_1: Seite kann nicht ausgeschlossen werden

- Algorithmus zur RNN-Suche

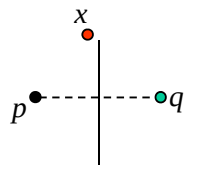
```
RdNN-Tree-Search(pa, q)          // pa = Diskadress z.B. der Wurzel des Indexes
  result = ∅;
  p := pa.loadPage();
  IF p.isDataPage() THEN
    FOR i=0 TO p.size() DO
      IF dist(q, p.getObject(i)) ≤ p.getNNDist(i) THEN
        result := result ∪ getObject(i);

  ELSE                           // p ist Directoryseite
    FOR i=0 TO p.size() DO
      IF MINDIST(q, p.getRegion(i)) ≤ p.getNNDist(i) THEN
        result := result ∪ RdNN-Tree-Search(p.childPage(i), q);
  RETURN result;
```

- Auch hier wieder alle möglichen anderen algorithmischen Lösungen zur NN-Suche anwendbar (Prioritätssuche, etc.)
- Vorteil
  - » Sehr gute Selektivität, damit gute Performanz bei Anfragen
  - » Seitenüberlappung wie bei „normalem“ R-Baum, daher kein extra Index für NN-/RQ-Anfragen nötig
- Nachteil
  - » *k* muss fest vorgegeben sein
  - » Nur für Vektordaten
  - » Weiterhin schlechte Performanz bei Einfügungen und Löschungen

– Geometrische RNN-Suche (Filter/Verfeinerung)

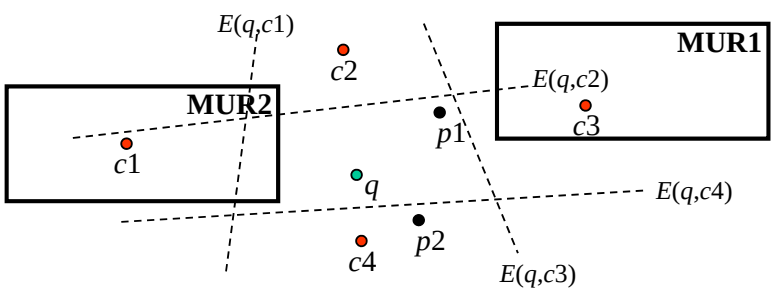
[Tao, Papadias, Lian. Int. Conf. Very Large Databases (VLDB), 2004]

- Idee
  - Gegeben: Voronoi-Ebene zwischen q und beliebigen Punkt p
  - Liegt ein Punkt x auf der Seite von p dieser Voronoi-Ebene, kann q nicht NN von x sein und damit $x \notin \text{RNN}(q)$
  - Voronoi-Ebene $E(p,q)$: für alle Punkte $e \in E$ gilt: dist(q,e) = dist(p,e)



Voronoi-Ebene $E(p,q)$ zwischen p und q

- Algorithmus: Filter-Schritt (Skizze)
  - Berechne ein NN-Ranking der DB
  - Solange noch Objekte im Ranking sind:
    - » Rufe getNext() auf
    - » Wenn aktueller Punkt p nicht „hinter“ einer Voronoi-Ebene liegt, konstruiere neue Voronoi-Ebene $E(p,q)$; p wird zur Kandidatenmenge hinzugefügt
    - » Punkte/Directoryseiten, die „hinter“ einer der Voronoi-Ebenen liegen (außer der eigenen), können aus dem Ranking/Kandidatenmenge gelöscht werden
  - Punkte, die die Ebenen bestimmen, müssen verfeinert werden, d.h. für diese Punkte muss jeweils eine NN-Anfrage berechnet werden
- Beispiel



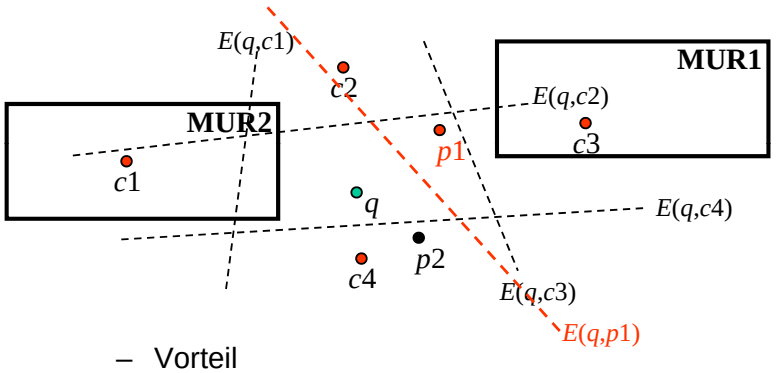
Bisherige Kandidaten:

{$c1$, $c2$, $c3$, $c4$}

Inhalt des Rankings (ungeordnet):

MUR1: nicht verfeinern
MUR2: verfeinern
$p1$: verfeinern
$p2$: nicht verfeinern

- Verfeinerung von $p1$
  - » Streiche $c2$ und $c3$ aus Kandidatenliste (liegt nun hinter $E(q,p1)$)
  - » MUR2 muss weiterhin verfeinert werden



Bisherige Kandidaten:
{$c1$, $c4$, $p1$}

Inhalt des Rankings (ungeordnet):
MUR2: verfeinern

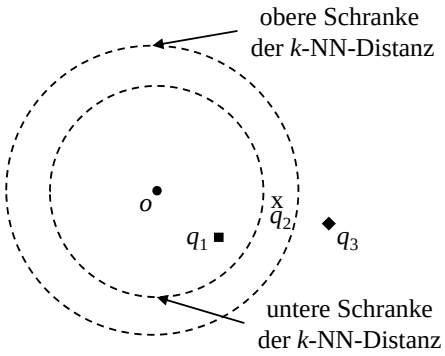
- Vorteil
  - » k kann beliebig sein (Ausschlusskriterium: Objekt/Seite muss hinter k Ebenen liegen
  - » Keine vorberechneten Distanzen, daher keine Update-Problematik und bessere Speicherkomplexität
- Nachteil
  - » Nur für Vektordaten
  - » Teurer Verfeinerungsschritt (eine NN-Anfrage pro Kandidat)
  - » Teilweise komplexe Ebenenverwaltung

– MRkNNCoP-Baum

[Achtert, Böhm, Kröger, Kunath, Pryakhin, Renz. ACM Int. Conf. Management of Data (SIGMOD), 2006]

- Vergleich bisheriger Verfahren
  - RNN-Tree/RdNN-Tree: Vorberechnung der NN-Distanz
    - » Wert für k ist fix und vorher bekannt
    - » Update-Problematik
    - » Dafür: erweiterbar auf metrische Daten (z.B. M-Tree)
  - Geometrische Suche
    - » Nur für Vektordaten
    - » Teurer Verfeinerungsschritt, schlechtere Selektivität
    - » Dafür: beliebiges k, keine Update-Problematik
- Idee:
  - Benutze die gute Selektivität der vorberechneten NN-Distanzen
  - Berechne für mehrere (am besten alle) Werte für k die k-NN-Distanzen vor
  - Problem: Speicherung aller Distanzen zu aufwendig
    - » Pro Objekt alle k-NN-Distanzen => Index wäre sehr hoch => hohe Kosten
  - Lösung: Approximiere die k-NN-Distanzen
    - » Approximation sollte untere Schranke (LB) der k-NN-Distanz sein => true drops, wir können Objekte (Seiten) frühzeitig ausschließen
    - » Zusätzliche Approximation als obere Schranke (UB) => true hits

- Bewertung von Objekt *o* (analog: Seiten) mit UB- und LB-Approximationen
  - dist(*o*,*q*) ≤ $LB_{k\text{-NN-Dist}}(o)$
    => *o* true hit, d.h. $o \in RNN(q, k)$
    - » Beispiel: $q = q_1$
  - dist(*o*,*q*) ≥ $UB_{k\text{-NN-Dist}}(o)$
    => *o* true drop, d.h. $o \notin RNN(q, k)$
    - » Beispiel: $q = q_2$
  - $UB_{k\text{-NN-Dist}}(o) \leq dist(o,q) \leq LB_{k\text{-NN-Dist}}(o)$
    => *o* Kandidat
    - » Beispiel: $q = q_2$



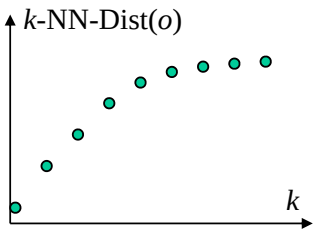
- Gegeben: für jedes Objekt *o* eine Sequenz der *k*-NN-Distanzen, <1-NN-Dist(*o*), 2-NN-Dist(*o*), …, k_{max}-NN-Dist(*o*)> für ein hinreichend großes k_{max}
- Frage: wie kann ich UB- und LB-Approximation dieser *k*-NN-Distanzen berechnen und kompakt speichern?

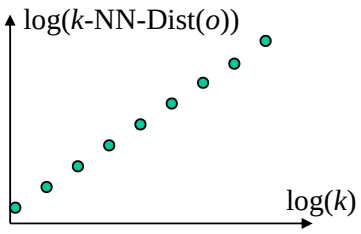
- Lösung aus der Theorie der Selbstähnlichkeit
  - Potentzgesetz gilt für Verhältnis zwischen
    - » dem Radius einer Hyperkugel
    - » der Anzahl an Objekten innerhalb der Hyperkugel

$$encl(\varepsilon) \propto \varepsilon^{d_f}$$

wobei *encl*(ε) = #Objekte innerhalb der Kugel
d_f = „Fraktale Dimension“

  - Übertragung auf k-NN-Sphäre:
    - » ε = *k*-NN-Distanz
    - » *encl*(ε) = *k*
  - Im log-log-Raum: $\log(k\text{-NN-Dist}(o)) \propto \frac{\log(k)}{d_f}$





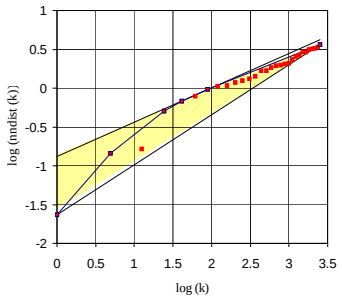
  - In der Realität verhalten sich die Distanzen nicht wie perfekte Linien im log-log-Raum
  - Trotzdem: im log-log-Raum können die *k*-NN-Distanzen mit einer Linie approximiert werden
  - Das ist erheblich billiger als alle *k*-NN-Distanzen zu speichern, oder andere Funktionen höherer Ordnung zu verwenden, um die Distanzen im normalen *k* / *k*-NN-Dist – Raum zu approximieren

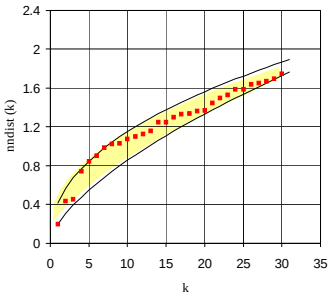
- LB- und UB-Approximationen
  - UB-Approximation ist eine Linie im log-log-Space, sodass

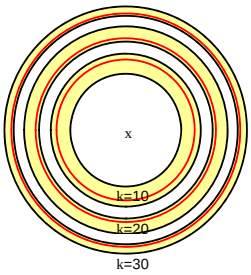
$$\forall k \le k_{\max} : k - \mathrm{NN} - \mathrm{Dist}(o) \le \mathrm{UB}_{k-\mathrm{NN}-\mathrm{Dist}}(o)$$

  - LB-Approximation ist eine Linie im log-log-Space, sodass

$$\forall k \le k_{\max} : k - \mathrm{NN} - \mathrm{Dist}(o) \ge \mathrm{LB}_{k-\mathrm{NN}-\mathrm{Dist}}(o)$$







- Jedem Objekt wird zugeordnet
  - Eine LB-Approximation der *k*-NN-Distanzen
  - Eine UB-Approximation der *k*-NN-Distanzen
- Jeder Seite im Index wird zugeordnet
  - Eine LB-Approximation der LB-Approximationen der Kindseiten
  - UB-Approximation wird nicht gespeichert, da zu wenig selektiv

- Vorteil:
  - Beliebiges *k*
  - Für allgemein metrische Daten (M-Tree) oder Vektordaten (z.B. X-Tree)
  - Durch UB- und LB-Approximationen höhere Filterselektivität als Geometrisches Verfahren => weniger Kandidaten die verfeinert werden müssen

- Nachteil
  - Updateproblematik
  - $k_{\max}$ muss bekannt sein (ABER i.d.R. kein Problem)
  - Teure Verfeinerung nötig (ABER i.d.R. deutlich weniger Kandidaten)

- ***Filter***-Algorithmus für allgemein metrische Daten (M-tree)

Knoten Node = (RoutingObj, CovRadius)
MINDIST(q,Node) = max{dist(q,Node.RoutingObj) – CovRadius, 0}

MRkNNCoP-Tree-Search(DB, q) // DB als MRkNNCoP-Tree organisiert
result = ∅;
candidates = ∅;
queue = **LIST OF** (dist:Real, obj:Object) **ORDERED BY** dist **ASCENDING**;
queue = [(0.0, DB.root)];
WHILE NOT queue.isEmpty() **DO**
p = queue.first().Object;
IF p.isDataPage() **THEN**
FOR i=0 **TO** p.size() **DO**
IF dist(q, p.getObject(i)) ≤ $LB_{k\text{-NN-Dist}}$(p.getObject(i)) **THEN**
result := result ∪ getObject(i);
ELSE IF dist(q, p.getObject(i)) ≤ $UB_{k\text{-NN-Dist}}$(p.getObject(i)) **THEN**
candidates := candidates ∪ getObject(i);
ELSE // p ist Directoryseite
FOR i=0 **TO** p.size() **DO**
IF MINDIST(q, p.getRegion(i)) ≤ $UB_{k\text{-NN-Dist}}$(p.getRegion(i)) **THEN**
queue.insert((MINDIST(q, p.getRegion(i)), p.childPage(i)));

– Zusammenfassung

Verfahren	Vorteile	Nachteile
RNN-Tree	Sehr gute Performanz, da keine Verfeinerung nötig	k fix; nur für Vektordaten; Updateproblematik; wenig selektiv bei normalen NN-Queries
RdNN-Tree	Sehr gute Performanz, da keine Verfeinerung nötig; auf allgemein metrische Daten erweiterbar	k fix; Updateproblematik
Geometrische (Voronoi-basierte) Suche	variables k;	Nur für Vektordaten; teure Verfeinerung nötig
MRkNNCoP-Tree	variables k; für allgemein metrische Daten	Verfeinerung nötig, Updateproblematik

2.6 Bewertung von Methoden zur Ähnlichkeitssuche

– Fragestellung
  - Anfragebearbeitung in metrischen Räumen oder Vektorräumen
  - Gesucht: Feature-Transformation zur Umwandlung komplexer STMM-Objekten in metrische Objekte/Featurevektoren
  - Wie gut drückt die Feature-Transformation die Ähnlichkeit der realen Objekte aus, d.h. wie gut approximiert die Distanz im Feature-Raum die Distanz im Objektraum?
  - Bewertung von Methoden zur Ähnlichkeitssuche („Ähnlichkeitsmodelle“)
    – Testset von Objekten
    – Stelle für alle Objekte des Testsets Ähnlichkeitsanfragen (typischerweise *k*-NN-Queries)
    – Evaluiere das Ergebnis dieser Anfragen

– Objekte mit bekannten Kategorien
  - Objekte sind in Kategorien eingeteilt und entsprechend markiert (z.B. „Schrauben“, „Nägel“, „Bolzen“, …), d.h. Ergebnis der Anfragen ist bekannt
  - Übersicht

	erwünscht	**unerwünscht**
gefunden	richtig positive (rp)	falsch positive (fp)
nicht gefunden	falsch negative (fn)	richtig negative (rn)

  - Recall (Sensitivität): Wie viele der erwünschten Objekte wurden gefunden?

$$\frac{rp}{rp+fn} = \frac{\text{gefundene erwünschte Objekte}}{\text{alle erwünschten Objekte}}$$

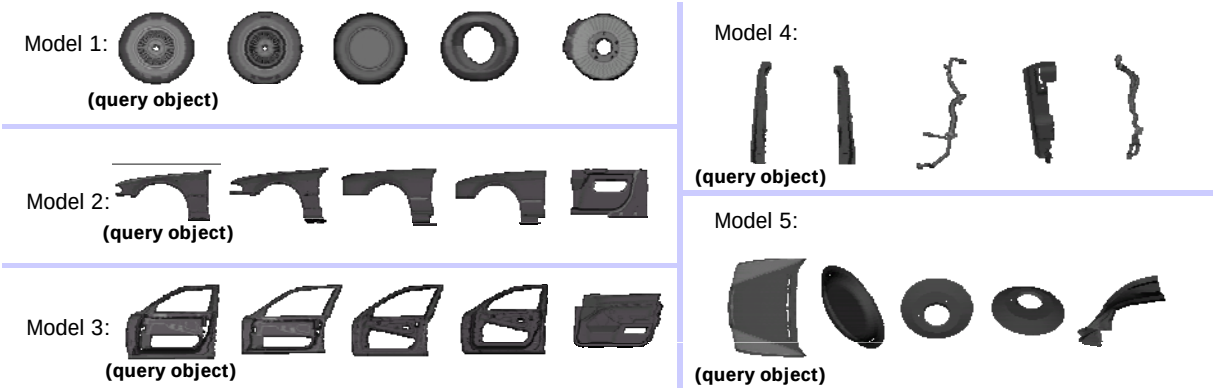
  - Precision: Wie viele der gefundenen Objekte sind erwünscht?

$$\frac{rp}{rp+fp} = \frac{\text{gefundene erwünschte Objekte}}{\text{alle gefundenen Objekte}}$$

  - Spezifität: WS, dass Test für unerwünschtes Obj. negativ verläuft

$$\frac{rn}{rn+fp} = \frac{\text{richtig negativ}}{\text{alle unerwünschten Objekte}}$$

– Objekte mit unbekannten Kategorien
  - Ergebnis der Anfragen ist unbekannt
  - Manuelle Evaluation weniger zufälligen *k*-NN-Queries
  - Problem: Qualität des Modells hängt ab von
    – einer geringen Anzahl von Query-Objekten
      » Besser: möglichst alle Objekte der DB spielen eine Rolle bei der Evaluation
    – der Wahl dieser Query-Objekte
      » Schlechtes Anfrageergebnis für gegebenes *q* bedingt nicht schlechtes Modell
      » Gutes Anfrageergebnis für gegebenes *q* bedingt nicht gutes Modell



- BOSS (Browsing OPTICS-plots for Similarity Search)
  [Brecheisen, Kriegel, Kröger, Pfeifle. Proc. SIAM Int. Conf. Data Mining (SDM), 2004]
  – Idee: benutze Data Mining Methoden
  – Clustering
    » Fasse Objekte in Gruppen zusammen, sodass die Objekte in einer Gruppe (Cluster) ähnlich, Objekte aus verschiedenen Clustern unähnlich sind
    » Hierarchisches Clustering: erstelle eine Hierarchie von ähnlichen Objekten
  – Clustererkennung und Clusterrepräsentation
    » Erkenne automatisch geeignete Cluster in der Hierarchie
    » Stelle jeden Cluster durch einen geeigneten Repräsentanten dar
  – Evaluation/Retrieval
    » Hierarchie von Clusterrepräsentanten ist navigierbar
    » Evaluation der Cluster um Ähnlichkeitsmodell zu evaluieren
    » Ähnlichkeits-basierte Suche nach Objekten ohne konkretes Anfrageobjekt angeben zu müssen